Harmonizer

Audrey Cooke, Michal Courson, Daniel Kohler, Allanah Matthews

Table of Contents

Introduction

The underlying purpose of any piece of musical technology is to both inspire artists to create new and interesting sounds, and to help them communicate with their audience. Our Harmonizer is an attempt to fulfill that purpose. This device is capable of pitch shifting any monophonic signal to a set of user-controlled musical notes. This allows the performer to create harmonies on the fly without the need for additional performers. The ability to harmonize with yourself in real time is a novel experience for most musicians, and thus can provide both inspiration and new possibilities for live performance.

There are a few devices similar to ours out in the world, but they are mostly custom built and tailored to the needs of specific artists. A few notable examples are Justin Vernon's instrument, "the Messina," which was created by Chris Messina, as well as Jacob Collier's harmonizer, built by MIT alumni Ben Bloomberg. Both devices are Frankenstein amalgamations of custom DSP processing code and commercially available hardware and software.[1][2] Our implementation, in contrast, uses only a Teensy 4.1 microprocessor to perform the required shifting operations. Our hope is that the simplicity of the Harmonizer makes this category of device more accessible to artists without a technical background.

High Level Overview

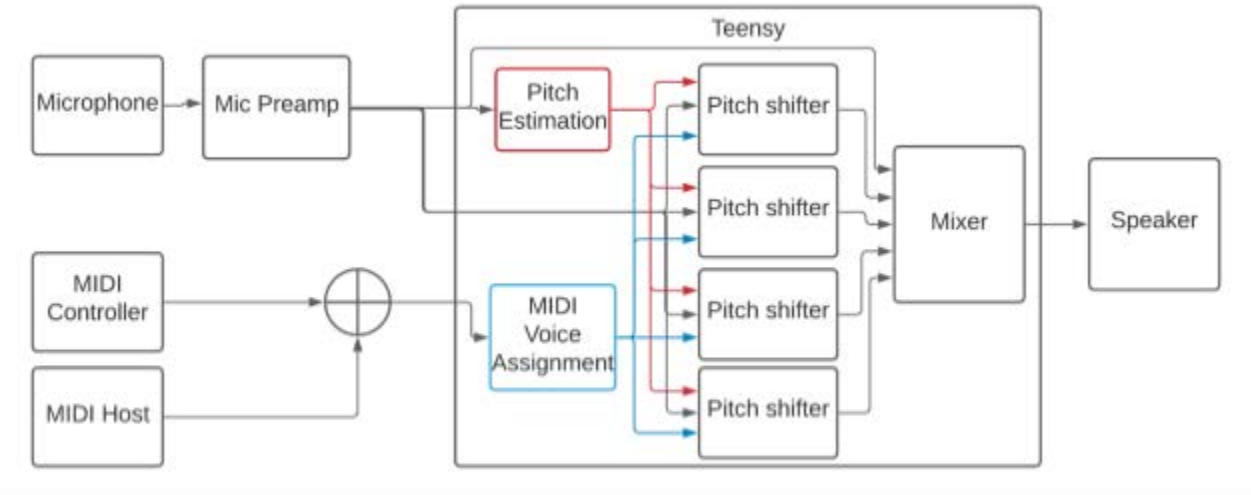

Figure 1: A high level block diagram of the harmonizer. Inputs include a balanced microphone and either a MIDI keyboard or a computer MIDI host. Output is line-level audio for integration with professional audio equipment.

Hardware

The Harmonizer is implemented in embedded C code on a PJRC Teensy 4.1 microcontroller. The Teensy is equipped with PJRC's Teensy Audio Shield, which implements an audio codec with a sample frequency of 44.1kHz. For increased compatibility with professional audio equipment, the device takes input directly from a balanced [6] microphone via an XLR cable and outputs to a ¼" patch cable.

To convert the balanced signal into unbalanced audio for the Audio Shield, the Harmonizer implements a simple microphone preamp. This preamp consists of an analog differential voltage converter, a second-order low pass filter with a cutoff frequency of 15kHz, and a microphone gain

stage controlled by a rotary potentiometer. The output of the conversion circuit is fed into the Line In audio port on the Audio Shield. Finally, the Audio Shield outputs the processed audio signal at line-level for performance or recording.

Powering the system is a ground shift circuit which provides ±5v rails to the op amps in the conversion circuit, and provides power to the teensyand USB devices. The harmonizer connects to either a computer via the Teensy's onboard micro USB port, or to an external MIDI keyboard via a USB-A host port. The schematics of these circuits may be found in Appendix A.

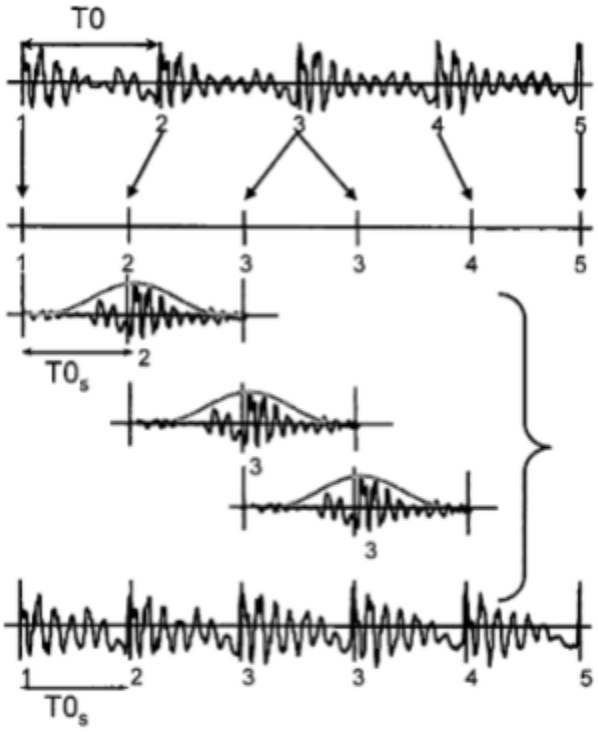

Figure 2: A visual representation of TD-PSOLA. Windows of the signal are overlapped and added to each other with new spacing to achieve pitch shifting [3].

TD-PSOLA

Time Domain Pitch Synchronous Overlap and Add (TD-PSOLA) is the primary algorithm that performs pitch shifting in the Harmonizer. The algorithm is explained in depth in [3]. TD-PSOLA performs pitch shifting by changing the spacing in between the peaks of each cycle of the input fundamental frequency. This is done generating "pitch markings" at each of these peaks, computing new "pitch markings" that are spaced according to the target fundamental frequency, and then overlapping windows of the original signal centered on the new pitch markings. Figure 2 is a visual representation of this process.

TD-PSOLA was chosen for two reasons. Firstly, the algorithm preserves the characteristic high-frequency components of the input, which helps the shifted audio to sound more natural. Furthermore, TD-PSOLA is very fast. Since this algorithm processes the signal exclusively in the time domain, it can be made extremely efficient.

Phase-Vocoding

Phase-Vocoding (PV) is the secondary algorithm we are using to pitch shift the input signal. The algorithm is implemented through an analysis-synthesis process. In the analysis stage, the FFT is taken on two consecutive windows of the input signal, separated by a constant hop size known as the "analysis hop." Phase and magnitude calculations are then performed on each respective bin of both FFTs in order to synthesize the new frequency domain version of the original signal. The inverse FFT is then taken on the synthesized result to get the signal into the time domain.

Overlap and add is performed on consecutive synthesized results, and the overlap amount is determined by the "synthesis hop". The analysis hop size to synthesis hop size ratio must be equivalent to the pitch shift ratio, or else the time stretched output will not be correct. The final step is to resample the time stretched version such that the pitch shift has been made. Figure 3 demonstrates this process [3].

Granular Synthesis

Granular Synthesis is an audio construction method that splits a signal into smaller pieces between 1-100ms known as grains [4]. Grains can be altered in frequency and intensity, may be arbitrarily re-ordered, and even layered on top of one another [4]. Altering the pitch of a grain requires stretching and resampling, which alters the duration of that grain. To change pitch independently of time, grains can be repeated or removed, known as looping and extracting respectively. A similar method can be used to change the time duration without affecting the pitch.

Grains are generally windowed to attenuate the beginning and end of the grain to facilitate a smooth transition between grains. Discontinuities on grain boundaries create a clicking noise that can be unpleasant [5]. In addition, leaving too much space between the grains causes noticeable gaps as if the signal was going through a fan. The length of grains and use of windowing can be manipulated to change the characteristics of the output of Granular Synthesis

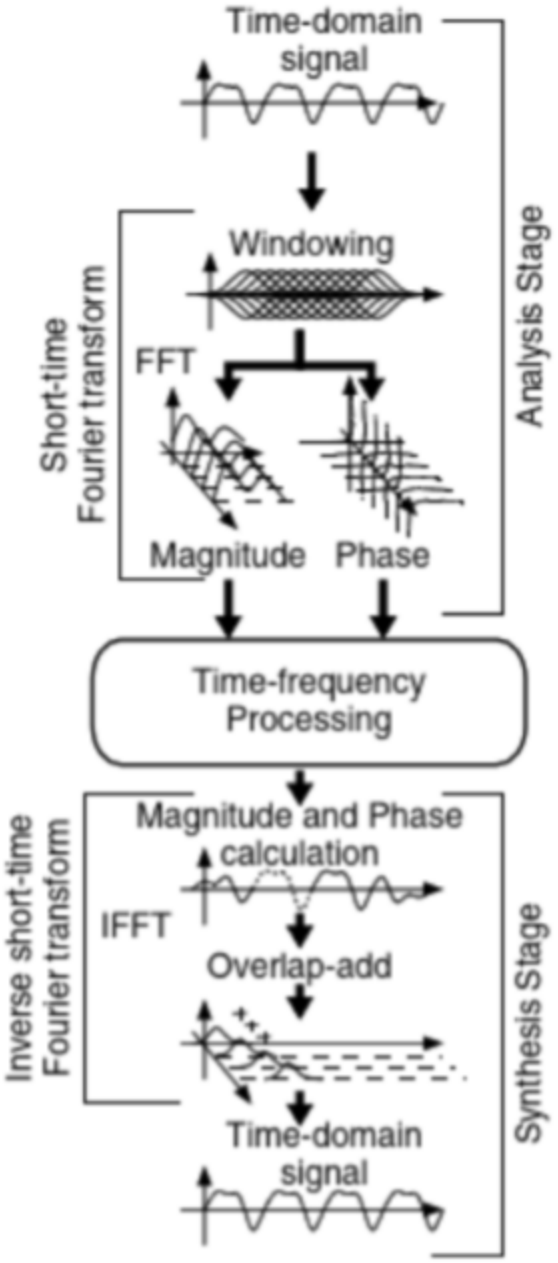

Figure 3: A visual representation of Phase Vocoding. Time stretching is done by calculating new phases, then resampling takes place. Image adapted from [3].

Technical Issues

TD-PSOLA

One of the trickiest things to get right when implementing TD-PSOLA for real-time is getting seamless transitions between audio blocks. These transitions from one block to another can happen at any spacing from the nearest pitch marking. This means that special care must be taken to track where the rollover from each block starts and ends. Getting this wrong causes instantaneous jumps in the audio, causing "popping" sounds in the output. A solution to this problem is to use a buffer to hold on to the rollover from the last processed block, making sure the first pitch marking for the output is spaced relative to the last pitch marking from the last block. This ensures consistent spacing over multiple blocks.

Another issue faced with TD-PSOLA relates to the lowest frequency that the algorithm could output. When calculating the size of the window applied to the input signal, there are checks to make sure that no array out of bounds errors occur. The window size is shrunk in the case where these errors could happen. This means that the combination of a large enough window size (a fundamental frequency with a large period) and an input buffer that is too small, the window size on the last iteration for a block could be very small, leading to a section at the beginning of the next block that is silent. The output therefore had a choppy quality to it, where there were occasional gaps of silence that lasted for a few milliseconds when the target pitch was a

low frequency (around 120 Hz and below). The issue was solved by delaying the output by a full block of 1024 samples so the algorithm had access to a whole block "in the future" and didn't need to adjust the window size as often.

Phase-Vocoding

The hardest part of Phase-Vocoding to get right is the synthesis phase calculations. Early iterations of the C++ offline implementation produced low quality output due to incorrect phase calculation for the negative half of the FFT array. Shifted signals suffered from unpleasant clicks and pops as the phases between consecutive synthesis windows were inconsistent. To fix this issue, phase calculations were only made for positive frequencies, and negative frequencies were set to the complex conjugate of the corresponding positive frequency. This resulted in smooth output.

Upon translation of the offline version into the realtime version, the primary issue encountered was that the algorithm could not process the input fast enough before the next input block came in. Initially, the Audio Library infrastructure was using a block size of 128 samples (about 2.9ms at 44.1 kH). Early online implementations of Phase-Vocoding took roughly 17ms to process. The delays induced by the slow speed of the algorithm resulted in unpleasant audio output. To fix this issue, the Teensy Audio Library infrastructure was modified to use a block size of 1024 instead of the default 128, changing the amount of time available to process a block to roughly 23.2 ms. However, this version was still too slow to run multiple concurrent pitch shifts. To fix this, the open source FFT algorithm used was swapped to the ARM-CMSIS RFFT implementation which is much more heavily optimized for the ARM Cortex processor on the Teensy. In doing so, processing time dropped to an average of roughly 5 ms per block, allowing for up to four concurrent shifts.

Granular Synthesis

The first issue faced while prototyping Granular Synthesis in Python was completely silent output. The WAV files produced consisted of data arrays filled with zeros. However, the output was the same length as the input. The issue resulted from a data type mismatch as floating point numbers from 0-1 were directly cast into 16 bit integers without any scaling.

Once the number format mismatch was fixed, a larger issue was revealed regarding the quality of output. Shifted audio sounded as if it were played through a fan, indicating discontinuities between the grains in the output. There was also a perceptible, consistent low frequency noise that seemed to be correlated with the grain size

With these two artifacts arising, Granular Synthesis failed our quantitative testing metrics as the pitch detection algorithms failed to recognize a fundamental frequency from the output. After much investigation, assistance from faculty, and significant debugging time, the cause of these artifacts could not be determined, so development of Granular Synthesis was not further implemented. A possible solution could be to use the Granular Synthesis method implemented in the Teensy Audio Library. However, the algorithm was developed to sound "glitchy" and did not fit the specifications of this project.

MIDI

The Harmonizer uses a class-compliant USB MIDI interface implemented in the USBHost_T36 Library [7]. The use of this interface allows the user to send MIDI signals from either a computer running a Digital Audio Workstation such as Ableton Live, or from a USB MIDI keyboard such as the

M-Audio Oxygen 49 digital piano. Due to timing constraints induced by Phase-Vocoding, it was decided that the Harmonizer would be programmed to shift no greater than four voices at a time. Since multiple keys can be played simultaneously on a keyboard, a method was needed to limit the number of active voices at a time. This goal was achieved through the use of a saturating circular buffer. The MIDI buffer adds each newly received MIDI note to a circular array, and evicts notes either upon saturation or note release.

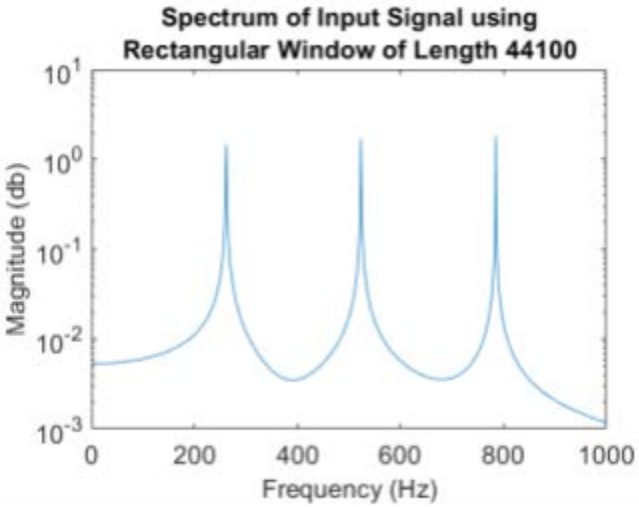

Figure 4: Frequency spectrum of tritonal sinusoidal input signal.

Testing

In order to quantitatively test the effectiveness of the Harmonizer's pitch shift algorithms, a tritonal sinusoid was fed into the device and shifted ± 2 octaves in semitone steps. A tritonal sinusoid signal was chosen for input due to the existence of harmonics and high frequency components in human voice and many monophonic instruments. A tritonal signal allows for analysis of the effects of different shifting techniques on frequency components other than those at the base frequency of a tone.

The tritonal sinusoidal signal is comprised of equal-magnitude harmonic components at f_0=261.63 Hz, f_1=2*f_0, and f_2=3*f_0. The signal was fed into the Harmonizer for a total of 49 seconds. In intervals of one second, the harmonizer was sent shift targets in the form of MIDI notes representing semitones from 65.41 Hz (C1) to 1046.50Hz (C5), a total of 49 individual notes spanning four octaves chromatically. Figure 4 shows the frequency spectrum of the input signal.

Results

Intonation

Truncated plots of the intonation of the tritonal sinusoid shifted using PV and TD-PSOLA are shown in Figure 5. In these plots, the y-axis is pitch (in semitones) and the x-axis is time. The thin orange line through the signal shows the pitch of the signal. When this line is in the middle of the row, it indicates that the signal is in tune.

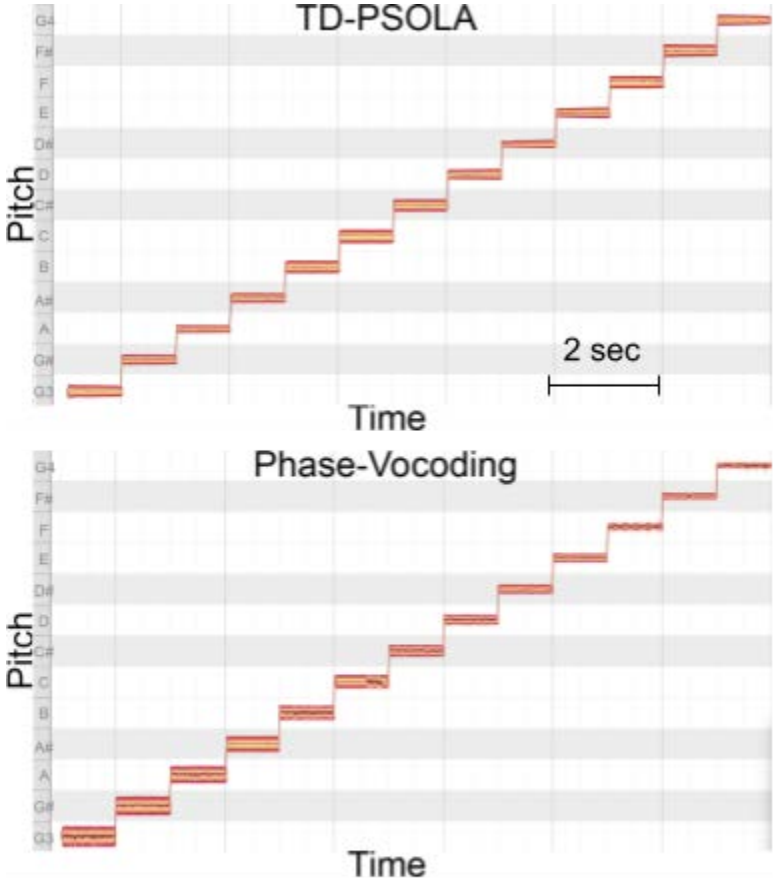

Figure 5: Tuning results for algorithm output for TD-PSOLA and Phase Vocoding with tritonal sinusoid as input, shifted chromatically from G3 (down a half octave) to G4 (up a half octave).

Frequency Performance

The STFTs of the Harmonizer output are plotted in Figure 6. STFTs use Chebychev windows of length Fs/4=11025 and an overlap of 0 for a total of 196 fourier transforms per spectrogram. As a control, tritonal signals were generated with f_0 from C1 to C5 with the same frequency ratios for f_1 and f_2 to match the shifted output. The STFT of this signal is compared against the outputs from each algorithm, as well as a method by which the algorithms are linearly crossfaded from PV to TD-PSOLA between a pitch shift ratio of 0.5 to 0.7.

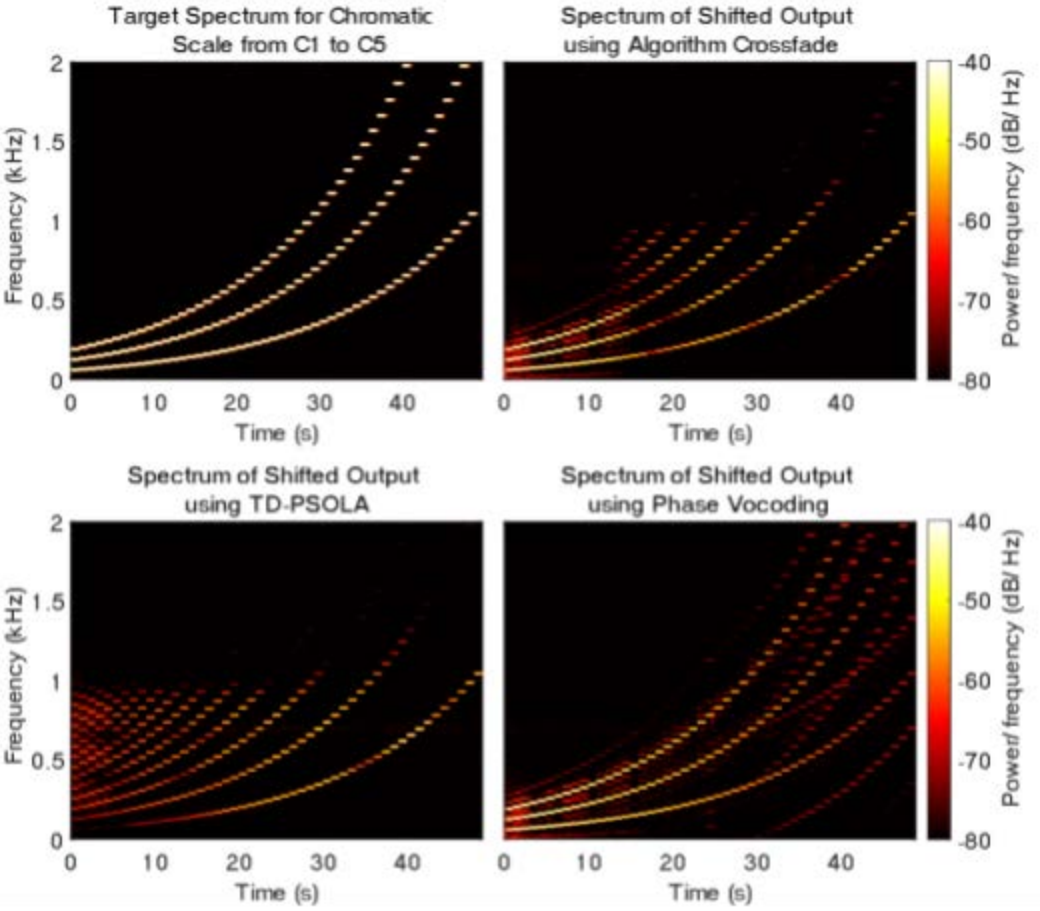

Figure 6: Spectrograms of synthesized target output frequency spectrum (top left), TD-PSOLA shifted signal (bottom left), Phase Vocoding shifted signal (bottom right), and crossfading method (top right).

Algorithm Speed

To measure speed of TD-PSOLA and Phase-Vocoding, the time to process one block of audio was measured and reported via the serial monitor in Arduino IDE. Figure 7 shows the results of these timing tests.

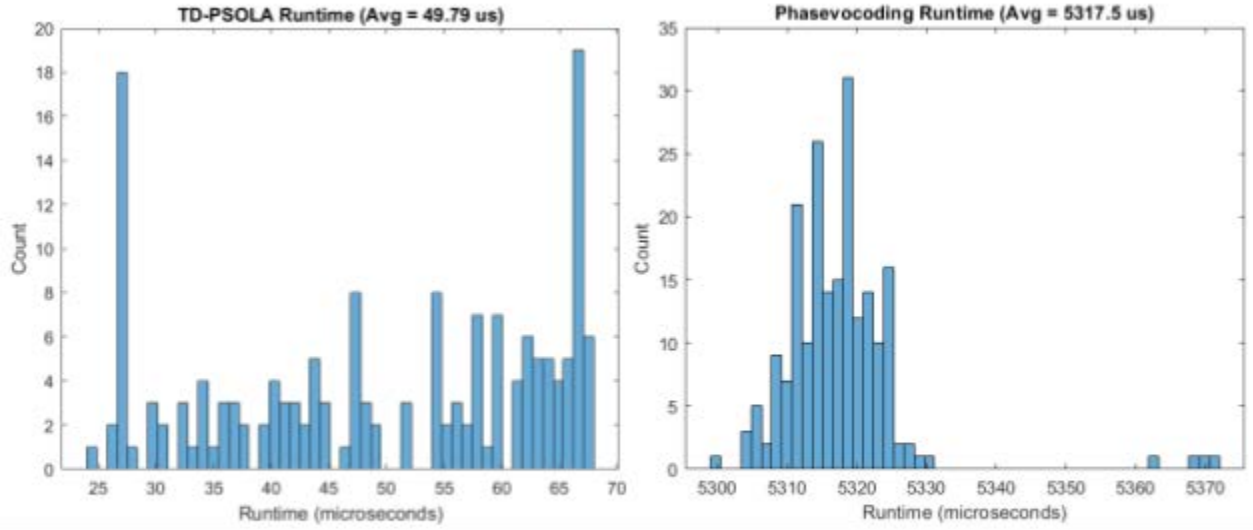

Figure 7: Timing test result histograms for TD-PSOLA and Phase Vocoding running in real time.

Hardware Verification

Hardware verification was performed using an oscilloscope to measure the steady state response of the preamp. The magnitude and phase of the response of the circuit are plotted in Figure 8.

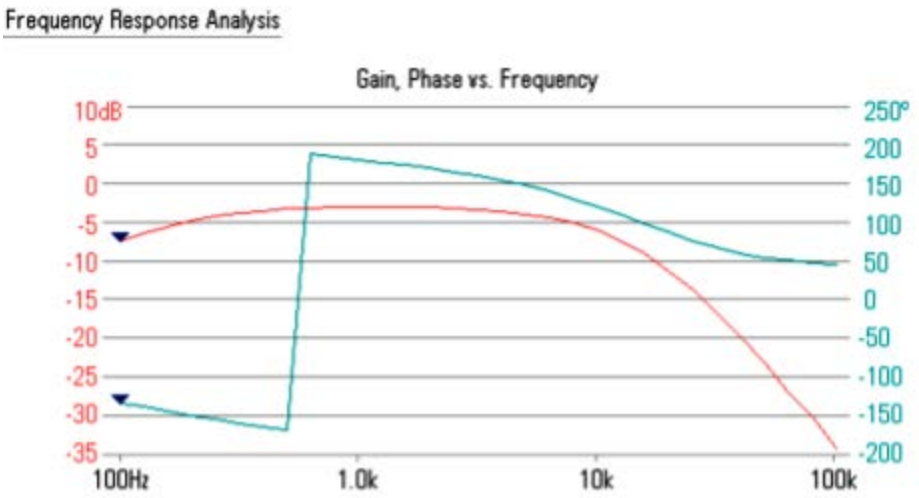

Figure 8: Steady-state frequency response of Microphone Preamp circuit with simulated balanced input, gain = 1.

Discussion

The tuning tests show that both Phase-Vocoding and TD-PSOLA are very good at staying in tune and hitting the target pitch. We know it's hitting the target pitch in tune as the graphs show the output signal in the dead center of each row.

Through our testing, we found that the subjective quality of the output from TD-PSOLA is very good in most cases. Shifting the pitch down by an octave or more produces audio that could be described as "thin" as the pitch markers become far enough apart that there are sections in the output that are silent. Coincidentally, this range of shift factors is where Phase-Vocoding sounds its best. The alterations of the formants in this range are not as distracting as they are with higher shifts, where the "talking while inhaling helium" effect kicks in.

These subjective findings were also backed up by the expanded STFT tests. Looking at figure 4, it is clear that TD-PSOLA has much more frequency artifacts and attenuation of the base frequency at low shift ratios, but has very few artifacts for higher shifts. Phase-Vocoding, while not perfect, is much closer to the desired spectrum in the low shift range and produces a lot of frequency artifacts for higher shifts. The STFT results of the crossfade clearly show that our solution produces output that is closer to the target spectrum than either of the two algorithms alone.

During testing, it was found that TD-PSOLA takes an average of about 50 microseconds, which is much faster than needed. The theoretical limits of how many voices TD-PSOLA could process concurrently is upwards of 300. Phase-Vocoding, on the other hand, takes an average of 5.3 milliseconds, which is *just* fast enough to run 4 times concurrently. Since Phase-Vocoding runs much slower than TD-PSOLA, our implementation only lets Phase-Vocoding run if it is needed for any particular voice. This is realized in the algorithm crossfade method, by which Phase-Vocoding will run for a pitch shift ratio of 0.7 or less. In the common case, this would limit the amount of voices using Phase-Vocoding concurrently to one or two voices, leaving plenty of breathing room where processing time is concerned.

During testing of the final product, it was noted that plugging a USB device into the USB host port on the Harmonizer, as well as plugging the harmonizer into a computer via USB, resulted in audible digital noise interference. This is a common occurrence among audio equipment, and is known as ground loop interference. In some cases, this type of interference can be rectified using a ground loop isolator, but it is unclear whether in the context of the Harmonizer this approach would be sufficient. The cause of and solutions to this issue were not explored during the development of the Harmonizer, as the interference had only a minor effect on the quality of the device’s output.

Finally, there is a small but noticeable amount of latency of around 46ms when quickly switching the input note frequency. This latency is due to the fact that 1024 samples must be analyzed to calculate the base frequency, and an additional 1024 samples must be processed to generate the shifted output. This is especially noticeable when using an instrument such as a saxophone or flute, but less apparent when singing into the device.

Future Work

Future versions of the Harmonizer could increase the number of concurrent voices. This would entail modifications to shifting algorithms to optimize their complexity. The biggest bottleneck on performance is the runtime of Phase-Vocoding, as demonstrated in Figure 7. Due to the complexity of FFT algorithms driving the approach, it is unclear whether or not further optimization could substantially improve performance of the algorithm. The most promising route for increasing voices would be to modify TD-PSOLA to incorporate some of the aspects of Phase-Vocoding for pitch shifts more than one octave below the base frequency. This could potentially extend the range of TD-PSOLA lower than the current implementation, allowing for PV to be omitted.

More experimentation can be performed with Phase-Vocoding to categorize the effects of different windows on the output sound. The only window tested in PV was a Hanning window, which is used in the current real-time implementation of the algorithm. Qualitative output comparisons should be made using different windows in order determine which window results in a more pleasant sounding output.

To minimize latency, older versions of the Harmonizer used an analysis sample size of 128. This resulted in a latency of around 25 ms when switching the input note. However, the code base had to be modified to support Phase-Vocoding, switching the sample size of 1024 globally This resulted in the discussed latency of 46ms. In order to rectify this issue, future versions of the Harmonizer could implement a sample accumulation method for collecting 1024 samples for shift processing while maintaining the Audio Library’s default sample size of 128. This approach was explored with early versions of the TD-PSOLA implementation, but will require further research and optimization to be a viable solution.

The inclusion of Granular Synthesis in the Harmonizer was cut early on in the project. In future versions, it may be worthwhile to explore other methods of pitch shifting than the two implemented in the final version of the code. Issues present in Granular Synthesis could potentially be solved through the use of a linear adaptive filter or similar technique to improve the quality of its output. Other possible shifting methods include time stretch and resampling via nearest-neighbor interpolation and low pass filtering, or a similar method using linear interpolation or spline interpolation.

The Harmonizer could also be improved by adding more elements to the signal chain within code. Adding extra controllable effects would allow the user to be much more expressive while performing with the device. Some ideas that have been discussed are filters, reverb, delay, portamento, pitch delta control, as well as the ability to "freeze" a chord in a secondary looping buffer and then continue to play extra notes over that "frozen" chord.

The circuitry of the Harmonizer could be further improved to provide for more efficient, less noisy power delivery to the active components and the microcontroller. In addition, the audio preamp could be redesigned to minimize the effects of USB noise and ground loop interference on the output signal. Though these changes are not necessary, they definitely improve the experience when performing with the Harmonizer.

References

[1] Feist, E. (2017, November 17). *How was Jacob Collier's harmonizer built?* FloVoice. Retrieved from https://www.flovoice.com/articles/6001397-how-was-jacob-colliers-harmonizer-built.

[2] Petrarca, E. (2016, December 19). *The sound engineer behind Bon Iver's "22, a million" clears up any confusion about its technical creation*. W Magazine. Retrieved from https://www.wmagazine.com/story/the-engineer-behind-bon-ivers-22-a-million-clears-up-any-confusion-about-its-high-tech-sound.

[3] Royer, T. (2019). *Pitch-shifting algorithm design and applications in music*. Retrieved from http://kth.diva-portal.org/smash/get/diva2:1381398/FULLTEXT01.pdf.

[4] Music Production Nerds (2021, January 7). *Granular synthesis explained*. Retrieved from https://musicproductionnerds.com/granular-synthesis-explained.

[5] Dobrian, C. *Windows and Envelopes.* Computer Music Programming. Retrieved from https://dobrian.github.io/cmp/topics/control-signals/4.Windows-and-Envelopes.html.

[6] B. Whitlock. (1995 June.) *Balanced Lines in Audio Systems: Fact, Fiction, and Transformers.* J. Audio Eng. Soc., vol. 43, no. 6, pp. 454-464,. Retrieved from http://davel.datatruck.com/Balanced%20Lines%20in%20Audio%20Systems%20-%20Bill%20Whitlock%20-%20OCR.pdf.

[7] Paul Stoffregen (2021). *USBHost_t36.* 2021. Github repository. Retrieved from https://github.com/PaulStoffregen/USBHost_t36.

Appendix A: Circuit Schematics

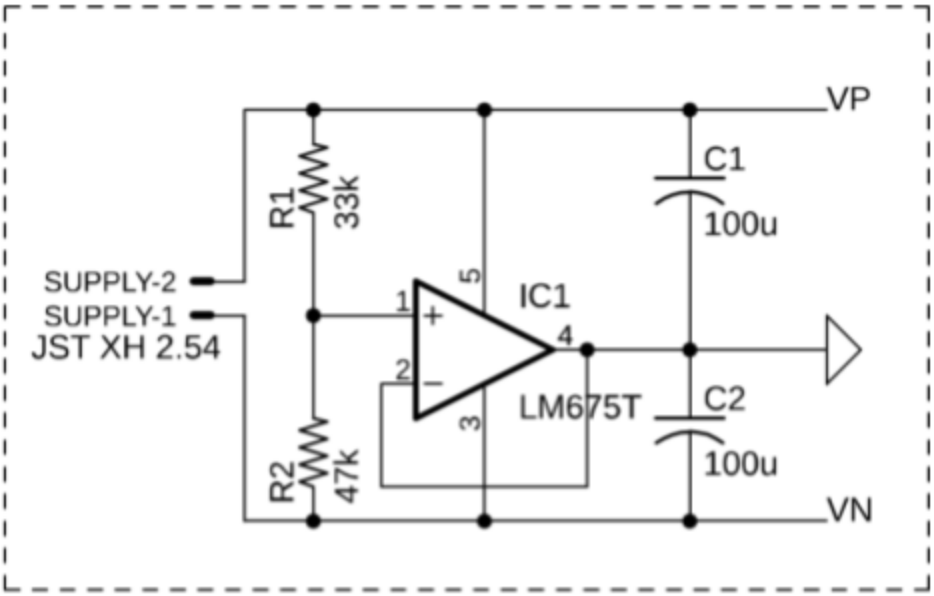

Figure A-1: Power circuit for powering preamp circuit and Teensy 4.1. Supplied is a 12v input voltage. The circuit uses a voltage divider with a unity-gain buffer to provide a 7v rail, which is assigned as the new ground. The power circuit outputs VP=5v and VN=-7v.

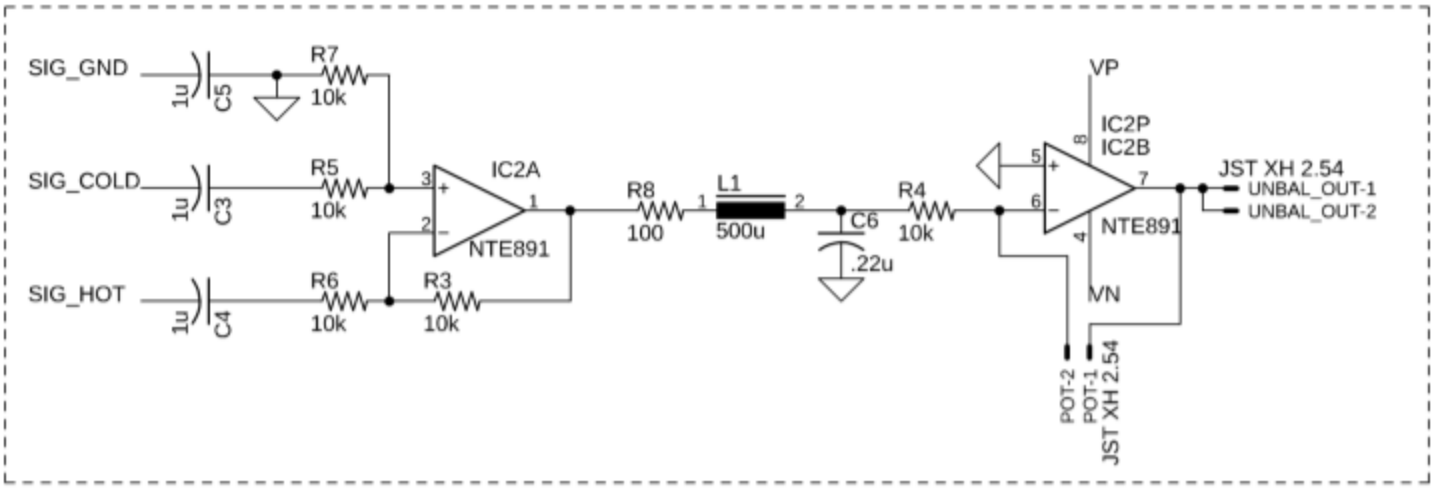

Figure A-2: Circuit schematic of the preamp circuit implemented for the Harmonizer. This circuit consists of the balanced audio input, a differential voltage converter using subtracting OpAmp circuit, a 15kHz second-order low pass filter, and finally a gain stage controlled by a potentiometer.